National Institute of Standards and Technology
U.S. Department of Commerce

NIST Interagency Report 7695
(DRAFT)

Common Platform Enumeration: Naming Specification Version 2.3 (DRAFT)

Brant A. Cheikes
David Waltermire

NIST Interagency Report 7695 (DRAFT)

Common Platform Enumeration: Naming Specification Version 2.3 (DRAFT)

Brant A. Cheikes
David Waltermire

C O M P U T E R   S E C U R I T Y

Computer Security Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930

August 2010

U.S. Department of Commerce

Gary Locke, Secretary

National Institute of Standards and Technology

Dr. Patrick D. Gallagher, Director

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analysis to advance the development and productive use of information technology. ITL's responsibilities include the development of technical, physical, administrative, and management standards and guidelines for the cost-effective security and privacy of sensitive unclassified information in Federal computer systems. This Interagency Report discusses ITL's research, guidance, and outreach efforts in computer security and its collaborative activities with industry, government, and academic organizations.

National Institute of Standards and Technology Interagency Report 7695 (DRAFT)
50 pages (August 2010)

Acknowledgments

The authors, Brant A. Cheikes of the MITRE Corporation and David Waltermire of NIST wish to thank their colleagues who reviewed drafts of this document and contributed to its technical content. The authors would like to acknowledge Harold Booth of NIST, Paul Cichonski of Booz Allen Hamilton, Seth Hanford of Cisco Systems, Inc., Tim Keanini of nCircle, Kent Landfield of McAfee, Inc., Mary Parmelee of the MITRE Corporation, Jim Ronayne of Cobham plc, and Shane Shaffer of G2, Inc. for their insights and support throughout the development of the document.

Abstract

Following security best practices is essential to maintaining the security and integrity of today's Information Technology (IT) systems and the data they store. Given the speed with which attackers discover and exploit new vulnerabilities, best practices need to be continuously refined and updated at least as fast as the attackers can operate. To meet this challenge, *security automation* has emerged as an advanced computer-security technology intended to help information system administrators assess, manage, maintain and upgrade the security posture of their IT infrastructures regardless of their enterprises' scale, organization and structure. The United States government, under the auspices of the National Institute of Standards and Technology (NIST), has established the Security Content Automation Protocol (SCAP—cf. scap.nist.gov) to foster the development and adoption of security automation standards and data resources.

The *Common Platform Enumeration* (CPE) addresses the security automation community's need for a standard method to identify and describe the software systems and hardware devices present in an enterprise's computing asset inventory. Four specification documents comprise the CPE stack:

1. Naming
2. Matching
3. Dictionary
4. Language

The Naming specification—this document—defines the logical structure of well-formed CPE names (WFNs), and the procedures for binding and unbinding WFNs to and from machine-readable encodings. The Matching specification defines the procedures for comparing WFNs to determine whether they refer to some or all of the same products or platforms. The Dictionary specification defines the concept of a dictionary of identifiers, and prescribes high-level rules for dictionary curators. The Language specification defines an approach for forming complex logical expressions out of WFNs. Collectively, the CPE specification stack aims to deliver these capabilities to the security automation community:

- A method for assigning unique machine-readable identifiers to certain classes of IT products and computing platforms;
- A method for curating (compiling and maintaining) dictionaries (repositories) of machine-readable product and platform identifiers;
- A method for constructing machine-readable referring expressions which can be mechanically compared (i.e., by a computer algorithm or procedure) to product/platform identifiers to determine whether the identifiers satisfy the expressions;
- A set of interoperability requirements which guarantee that heterogeneous tools can select and use the same unique identifiers to refer to the associated products and platforms.

Audience

This specification document defines standardized data models and machine encodings for creating product descriptions and identifiers. These models and encodings are envisaged to be of interest to the following audiences:

a. **Asset inventory tool developers.** Asset inventory tools inspect computing devices and assemble catalogs that list installed component hardware and software elements. In the absence of CPE, there is no standardized means for how these tools should report what they find. The CPE specification stack provides all the technical elements needed to comprise such a capability. Furthermore, CPE is intended to address the needs of asset inventory tool developers regardless of whether the tools have credentialed (authenticated) access to the computing devices subject to inventory.
b. **Security content automation tool developers.** Many security content automation tools are fundamentally concerned with making fully- or partially-automated information system security decisions based on collected information about installed products. The CPE specification stack provides a framework that supports correlation of information about identical products installed across the enterprise, and association of vulnerability, configuration, remediation and other security-policy information with information about installed products.
c. **Security content authors.** Security content authors are concerned with creating machine-interpretable documents that define organizational policies and procedures pertaining to information systems security, management and enforcement. Often there is a need to tag guidance, policy, etc., documents with information about the product(s) to which the guidance, policy, etc., applies. These tags are called *applicability statements*. The CPE specification stack provides a standardized mechanism for creating applicability statements which can be used to ensure that guidance is invoked as needed when the product(s) to which it applies is discovered to be installed within an enterprise.

Table of Contents

List of Figures and Tables

1. Introduction

1.1 Purpose and Scope

Following security best practices is essential to maintaining the security and integrity of today's Information Technology (IT) systems and the data they store. Given the speed with which attackers discover and exploit new vulnerabilities, best practices need to be continuously refined and updated at least as fast as the attackers can operate. To meet this challenge, *security automation* has emerged as an advanced computer-security technology intended to help information system administrators assess, manage, maintain and upgrade the security posture of their IT infrastructures regardless of their enterprises' scale, organization and structure. The United States government, under the auspices of the National Institute of Standards and Technology (NIST), has established the Security Content Automation Protocol (SCAP—cf. scap.nist.gov) to foster the development and adoption of security automation specifications and data resources.[1]

The foundation of an effective security automation system is the capability to completely and unambiguously characterize the software systems, hardware devices and network connections which comprise an enterprise's computing infrastructure. With a detailed computing asset inventory in hand, one can begin to integrate and correlate a wealth of other knowledge about, e.g., vulnerabilities and exposures,[2] configuration issues and best-practice configurations,[3] security checklists,[4] impact metrics,[5] and more.

The *Common Platform Enumeration* (CPE) addresses the security automation community's need for a standardized method to identify and describe the software systems and hardware devices present in an enterprise's computing asset inventory. Four specification documents comprise the CPE stack:

1. Naming
2. Matching
3. Dictionary
4. Language

The Naming specification—this document—defines the logical structure of well-formed CPE names (WFNs), and the procedures for binding and unbinding WFNs to and from machine-readable encodings. The Matching specification defines the procedures for comparing WFNs to determine whether they refer to some or all of the same products or platforms. The Dictionary specification defines the concept of a dictionary of identifiers, and prescribes high-level rules for dictionary curators. The Language specification defines a standardized structure for forming complex logical expressions out of WFNs. These four specifications are arranged in a *specification stack* as depicted in Figure 1-1. Henceforward we will refer to this stack as the *CPE specification stack*, and we will refer to the four-document set of specifications as the *CPE specification suite*.

[1] For more information on SCAP, cf. NIST Special Publication 800-117, *Guide to Adopting and Using the Security Content Automation Protocol*, http://csrc.nist.gov/publications/drafts/800-117/draft-sp800-117.pdf.
[2] See, e.g., MITRE's Common Vulnerabilities and Exposures (CVE) project, on the web at cve.mitre.org.
[3] See, e.g., MITRE's Common Configuration Enumeration (CCE) project, on the web at cce.mitre.org, and also the Federal Desktop Core Configuration (FDCC), on the web at fdcc.nist.gov.
[4] See, e.g., the National Checklist Program Repository, on the web at checklists.nist.gov.
[5] See, e.g., the Common Vulnerability Scoring System, on the web at nvd.nist.gov/cvss.cfm.

Figure 1-1: CPE Specification Stack

Collectively, the CPE specification stack aims to deliver these capabilities to the security automation community:

- A method for assigning unique machine-readable identifiers to certain classes of IT products and computing platforms;
- A method for curating (compiling and maintaining) dictionaries (repositories) of machine-readable product and platform identifiers;
- A method for constructing machine-readable referring expressions which can be mechanically compared (i.e., by a computer algorithm or procedure) to product/platform identifiers to determine whether the identifiers satisfy the expressions;
- A set of interoperability requirements which guarantee that heterogeneous security automation tools can select and use the same unique identifiers to refer to the associated products and platforms.

1.2 Scope

The CPE Naming Specification defines the concepts of *description* and *identification* (cf. Section 1.2.1), and applies these concepts types of computing products:

1. Applications (cf. Section 2.1.1)
2. Operating systems (cf. Section 2.1.9)
3. Hardware devices (cf. Section 2.1.8)

The CPE Naming Specification is concerned solely with describing and identifying product *classes* rather than product *instances* (cf. Section 1.2.2).

1.2.1 Description vs. Identification

The primary purpose of this specification is to provide a standardized framework for distinguishing information that *identifies* an individual product from information that merely *describes* a (possibly empty) set of products. In general terms, when one *describes* an entity in some domain of reference, one enumerates a set of attributes and their values possessed by that entity, for the purpose of helping a consumer of that description to distinguish that entity from other entities in the domain. For example, Joe might describe his car as a “2004 Subaru Outback with a black leather interior”. Conceptually, this description could be modeled as a set of attribute=value pairs, e.g.,

[year=2004, maker=subaru, model=outback, interior_color=black, interior_material=leather]

A description is said to be *ambiguous* relative to a defined universe of entities when the description is insufficient to enable an interpreter to distinguish a unique entity in the universe possessing all specified attributes and values. The above description is ambiguous relative to the universe of, e.g., all automobiles registered in the state of Massachusetts, but might not be ambiguous given a more narrowly defined universe (e.g., all automobiles registered in a particular nine-digit postal code region). To *identify* an entity is to uniquely describe it, and while under some circumstances a description may also be an identifier, an *identifier* is typically a symbol (alphanumeric or graphic) which serves as an index for picking a unique individual out of a universe of individuals.

The scope of the CPE Naming Specification encompasses description as well as identification. The specification describes a standardized method for forming (possibly ambiguous) descriptions of applications, operating systems, and hardware devices, as well as identifiers for applications, operating systems, and hardware devices.

1.2.2 Class vs. Instance

When describing or identifying applications, operating systems, and hardware devices, the CPE Naming Specification addresses only the description or identification of product *classes* rather than product *instances.* A "product instance" is a unique, physically discernable entity in the world—such as a specific licensed and configured installation of a product on a particular computing device owned by XYZ Corp. and physically installed in a particular location in the world. A "product class" is a set-theoretic abstraction over product instances. For example, one might say that the computing device owned by XYZ Corp. is a member of the class of computing devices known as "Lenovo ThinkPad X61".

Classes may be defined at varying levels of abstraction, e.g., "all computing devices manufactured by Lenovo", "all laptops manufactured by Lenovo", "all ThinkPads manufactured by Lenovo", etc. The CPE Naming Specification leaves all decisions about what constitutes useful or needed abstractions to the users. The Naming Specification takes the view that all names constitute descriptions of product classes, and the degree of abstraction of the description varies in proportion to the quantity of attribute-value pairs specified. A description is more concrete (less abstract) to the extent that it contains more attribute-value pairs, and less concrete (more abstract) to the extent that it contains fewer attribute-value pairs.

A description becomes an identifier relative to a defined universe of individuals when the description contains sufficient information to select a single individual from the universe.

1.2.3 Out of Scope

The following aspects of description and naming are outside the scope of the CPE Naming Specification:

- Representing relationships (e.g., part-of, bundled-with, released-before/after, same-as) between products described or identified;
- Representing user-defined configurations of installed products;
- Representing entitlement/licensing information about products;
- Defining procedures and guidelines for assigning "correct" or "valid" values to attributes of product descriptions or identifiers;
- Defining procedures and guidelines for creating or maintaining valid-values lists.

1.3 Normative References

The following documents are indispensible references for understanding the application of this specification.

[CPE22] Buttner, A. and N. Ziring. (2009). *Common Platform Enumeration—Specification*. Version 2.2 dated 11 March 2009. See: http://cpe.mitre.org/specification/spec_archive.html.

[ISO19770-2] ISO/IEC 19770-2. (2009). *Software Identification Tag.* November 2009. See: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53670.

[RFC2119] Bradner, S. (1997). *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. See http://www.ietf.org/rfc/rfc2119.txt.

[RFC2234] Crocker, D. and P. Overell. (1997). *Augmented BNF for Syntax Specifications: ABNF*. Internet RFC 2234, November 1997. See: http://www.ietf.org/rfc/rfc2234.txt.

[RFC3986] Berners-Lee, T., Fielding, R. and L. Masinger. (2005). *Uniform Resource Identifier (URI): Generic Syntax.* Internet RFC 3986, January 2005. See: http://www.ietf.org/rfc/rfc3986.txt.

[RFC4646] Phillips, A. and M. Davis. (2006). *Tags for Identifying Languages.* RFC 4646, September 2006. See: http://www.ietf.org/rfc/rfc4646.txt.

[SCAP800-117] NIST Special Publication 800-117, *Guide to Adopting and Using the Security Content Automation Protocol*. See: http://csrc.nist.gov/publications/drafts/800-117/draft-sp800-117.pdf.

[TUCA] *Common Platform Enumeration (CPE) Technical Use Case Analysis.* White Paper, The MITRE Corporation, November 2008. See: http://cpe.mitre.org/about/use_cases.html.

1.4 Document Structure

This specification document is organized as follows:

- Section 2 defines the key terms and abbreviations used herein;
- Section 3 defines what it means for an implementation or organization to conform with this specification;
- Section 4 places this specification in the context of related specifications and standards;
- Section 5 defines the data model of *well-formed CPE names*;
- Section 6 defines the procedures for *binding* and *unbinding* well-formed names into and out of formats suitable for machine interchange and processing;
- Section 7 defines the procedures for converting between bound forms;
- Appendix A provides informational notes on intended use cases;
- Appendix B documents per-release changes to this specification over time.

1.5 Document Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

Text intended to represent computing system input, output, or algorithmic processing is presented in `fixed-width Courier font`.

Normative references are listed in Section 1.3 of this document. The following reference citation conventions are used in the text of this document:

- For normative references, a square bracket notation containing an abbreviation of the overall reference citation, followed by a colon and subsection citation where applicable (e.g. [CPE-N:5.2.1] is a citation for CPE Naming specification, Section 5.2.1);
- For references within this document (internal references) and non-normative references, a parenthetical notation containing the “cf.” (compare) abbreviation followed by a section number for internal references or an external reference, (e.g. (cf. 2.1.4) is a citation for Section 2.1.4 of this document).

2. Terms, Definitions and Abbreviations

This section defines a set of common terms used within the document. Many terms have been imported from Section 4 of [ISO19770-2]. These are indicated by appending the particular subsection citation to the overall reference citation separated by a colon, e.g., [ISO19770-2:4.1.1].

2.1 Terms and Definitions

2.1.1 Application

An *application* is a system for collecting, saving, processing, and presenting data by means of a computer [ISO19770-2:4.1.1].

Notes:

- The term *application* is generally used when referring to a component of software that can be executed.
- The term *application* and *software application* are often used synonymously.

2.1.2 Asset Inventory Tool

An *asset inventory tool* is an application which runs within an enterprise's computing infrastructure and enumerates the computing devices and products comprising that infrastructure.

2.1.3 Bind

To *bind* means to connect two things together. In the context of this specification, to *bind* means to deterministically transform a logical construct into a machine-readable representation suitable for machine interchange and processing. The result of this transformation is called a *binding*. A binding may also be referred to as the "bound form" of its associated logical construct.

2.1.4 Bundle

A *bundle* is a grouping of products which is the result of a marketing/licensing strategy to sell use rights to multiple products as one purchased item [ISO19770-2:4.1.2].

Note:

A bundle can be referred to as a "suite", if the products are closely related and typically integrated (such as an office suite containing a spreadsheet, word processor, presentation and other related items).

2.1.5 Component

A *component* is an entity with discrete structure, such as an assembly or software module, within a system considered at a particular level of analysis [ISO19770-2:4.1.3].

Note:

Component refers to a part of a whole, such as a component of a software product, a component of a software identification tag, etc.

2.1.6 Computing Device

A *computing device* is a functional unit that can perform substantial computations, including numerous arithmetic operations and logic operations without human intervention [ISO19770-2:4.1.4].

Note:

A computing device can consist of a stand-alone unit, or several interconnected units. It can also be a device that provides a specific set of functions, such as a phone or a personal organizer, or more general functions such as a laptop or desktop computer.

2.1.7 Configuration Item

A *configuration item* is an item or aggregation of hardware or software or both that is designed to be managed as a single entity [ISO19770-2:4.1.5].

Note:

Configuration items may vary widely in complexity, size and type, ranging from an entire system including all hardware, software and documentation, to a single module, a minor hardware component or a single software package.

2.1.8 Hardware Device

A *hardware device* is a discrete physical component of an information technology system or infrastructure. A hardware device may or may not be a computing device (e.g., a network hub, a webcam, a keyboard, a mouse).

2.1.9 Operating System

An *operating system* is the software on a computing device that manages the way different applications use its hardware, and regulates the ways that users control the computer [Wikipedia].

2.1.10 Platform

A *platform* is a computer or hardware device and/or associated operating system, or a virtual environment, on which software can be installed or run [ISO19770-2:4.1.17].

Note:

Examples of platforms include Linux™, Microsoft Vista®, and Java.

2.1.11 Product

A *product* is a complete set of computer programs, procedures and associated documentation and data designed for delivery to a software consumer [ISO19770-2:4.1.19].

Note:

The terms “product” and “software package” are used interchangeably depending on the context of the item described.

2.1.12 Release

A *release* is a collection of new and/or changed configuration items which are tested and introduced into a production environment together [ISO19770-2:4.1.21].

2.1.13 Software

Software is all or part of the programs, procedures, rules, and associated documentation of an information processing system [ISO19770-2:4.1.25].

2.1.14 Software Creator

A software creator is a person or organization that creates a software product or package [ISO19770-2:4.1.28].

Note:

This entity might or might not own the rights to sell or distribute the software.

2.1.15 Software Manufacturer

A *software manufacturer* is a group of people or an organization that develops software, typically for distribution and use by other people or organizations [ISO19770-2:4.1.34].

2.1.16 Software Package

A *software package* is a complete and documented set of programs supplied for a specific application or function [ISO19770-2:4.1.35].

Notes:

- In the context of the CPE Naming Specification, the term software package refers to the set of files associated with a specific set of business functionality that can be installed on a computing device and has a set of specific licensing requirements.
- The terms “product” and “software package” may be used synonymously depending on the context of the item described.

2.1.17 Unbind

In general terms, to *unbind* means to disconnect two things from one another. In the context of this specification, to *unbind* means to deterministically transform a binding into its logical-form construct.

2.1.18 Uniform Resource Identifier

A *Uniform Resource Identifier* (URI) is a compact sequence of characters that identifies an abstract or physical resource available on the Internet.

Note:

The syntax used for URIs is defined in [RFC3986].

2.2 Abbreviated Terms

CPE	Common Platform Enumeration
IT	Information Technology
NIST	National Institute of Standards and Technology
SCAP	Security Content Automation Protocol
WFN	Well-formed Name
URI	Uniform Resource Identifier

3. Conformance

Products may want to claim conformance with this specification for a variety of reasons. This section provides the high-level requirements that must be met by any implementation seeking to claim conformance with this specification.

Implementations conforming to this specification MUST:

1. Make an explicit claim of conformance to this specification in any documentation provided to end users.
2. Produce and/or consume syntactically correct Formatted String bindings as needed to describe or identify applications, operating systems and hardware devices (cf. 6.3).

In addition, if the implementation is a consumer of CPE names, to claim conformance to this specification it SHOULD be able to consume (i.e., accept as valid input) any CPE name that meets the requirements specified in [CPE22], and, if necessary, to convert that CPE name to a syntactically correct Formatted String binding (cf. 7.1).

These requirements are intended to guarantee that a conformant implementation not only can produce and/or consume the newly-introduced Formatted String binding form as needed to interoperate with other implementations, but also to process legacy product identifiers as well.

For implementations conforming to this specification it is OPTIONAL that they be able to convert any syntactically correct Formatted String binding to a valid CPE name that meets the requirements specified in [CPE22] (cf. 7.2). This optional feature may enable a conforming implementation to interoperate to a limited extent with implementations conforming to [CPE22] and possibly prior releases as well.

4. Relationship to Existing Specifications and Standards

This section is informative in nature, and is intended to characterize the relationship between this specification and any related specifications or standards (both current and past).

4.1 Relationship to CPE v2.2

The CPE specification suite is intended to replace [CPE22]. Whereas [CPE22] defined all elements of the CPE specification in a single document, starting with this release the design has been changed to a stack model. In the stack model, capabilities are built incrementally out of simpler, more narrowly defined elements that are specified lower in the stack. This design opens opportunities for innovation, as novel capabilities can be defined by combining only the needed elements, and the impacts of change can be better compartmentalized and managed. The CPE specification stack and specification suite are intended to provide all the capabilities made available by [CPE22] while adding new features suggested by the CPE user community.

4.2 Relationship to ISO/IEC 19770-2

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) have published ISO/IEC-19770 Part 2, "Software Identification Tag". As explained in the introduction to the standard,

> *The software identification tag is an XML file containing authoritative identification and management information about a software product. The software identification tag is installed and managed on a computing device together with the software product. The tag may be created as part of the installation process, or added later for software already installed without tags. However, it is expected more commonly that the tag will be created when the software product is originally developed, and then be distributed and installed together with the software product.* [ISO19770-2], p. vi.

Both the CPE *specification stack* and ISO/IEC-19770-2 address the need to standardize the way products are identified. CPE differs, however, in a number of respects:

- The scope of CPE is somewhat broader, including hardware devices as well as software, and distinguishes operating systems from general software applications;
- CPE emphasizes the development and use of "common identifiers" enabling a wide variety of information about the same product or class of products to be correlated;
- CPE provides support for the creation of product descriptions as well as product identifiers.

There are also many areas in which the two efforts overlap or complement one another. Published in November 2009, ISO/IEC 19770-2 is a relatively new standard that is in the process of raising industry awareness and building its user base. As such, we expect that the similarities and differences between the two efforts will become increasingly evident as both continue to mature.

5. Data Model Overview

This section defines the foundational logical construct of the CPE specification suite—the *well-formed CPE name,* abbreviated WFN.

5.1 Motivation

[CPE22] defines the CPE name as a multi-component URI obeying a specified grammar. The present specification departs significantly from that practice by first introducing a logical construct—the *well-formed name* (WFN)—then defining procedures for *binding* and *unbinding* this construct to and from machine-readable representations.

The principal motivation in doing so was to create opportunities for future growth and innovation in the ways in which machines exchange product descriptions. During the development of this specification a clear need was recognized to define at least two different machine-readable representations (sometimes called “transports”) for product descriptions, one for backward compatibility with prior releases of the CPE specifications, and a second to provide critical new features demanded by the user community. As work advanced, community members proposed additional transport representations for consideration. As the inventory of potential representations increased, it became clear that there could be serious challenges involved in defining numerous conversions among transports and procedures for pair-wise comparison. Consequently, an abstract canonical form—a kind of interlingua—was chosen to serve as the standardized form for processing CPE information.

Using this interlingua it is possible to define conversions simply in terms of transforms into and out of the canonical form, and define matching and other higher-level processes in generic rather than representation-specific terms. The WFN form specified below lays the foundation for new binding forms to be introduced in the future without affecting other specifications defined in terms of the canonical form.

5.2 Definitions and Notation

Section 5.2.1 defines the *well-formed CPE name* (WFN). Section 5.2.2 describes the notation convention used in this specification document for illustrating WFNs.

5.2.1 Well-Formed CPE Name

A *well-formed CPE name* (WFN) is defined to be an unordered set of attribute-value pairs that collectively (a) describe or identify a software application, operating system, or hardware device, and (b) satisfy the criteria specified in Section 5.3. *Unordered* means that there is no prescribed order in which attribute-value pairs must be listed, and there is no specified relationship (hierarchical, set-theoretic or otherwise) among attributes or attribute-value pairs.

The WFN is a logical construct only. The WFN is not intended to be a data format, encoding, or any other kind of machine-readable representation for machine interchange and processing. Rather, it is a conceptual data structure—an abstract canonical form—used here for the purpose of clearly and unambiguously specifying desired implementations and behaviors. There is no requirement that CPE-conformant tools create or manipulate WFN-like data structures internally to their implementations. Section 6 describes procedures for *binding* WFNs to machine-readable representations for interchange and processing.

An *attribute-value pair* is a tuple *a*=*v* in which *a* (the *attribute*) is an alphanumeric label (used to represent, e.g., a property or state of some entity), and *v* (the *value*) is the value assigned to the attribute. Lexical case SHALL NOT distinguish attributes from one another, e.g., the attributes Foo, foo, FOO, etc., SHALL be considered equivalent. By convention, attributes will be written in all lowercase letters, with the underscore ("_") character used to separate distinct words within an attribute.

The following are examples of attribute-value pairs:

- color=red
- vehicle_length=6
- unit=meter
- nickname="Zippy"

5.2.2 Notation

When illustrating WFNs in this document the following notation will be used:

```
wfn:[a1=v1, a2=v2, …, an=vn]
```

That is, WFNs will be notated as *lists of attribute-value pairs enclosed in square brackets, prefixed with the string "wfn:"* This notation is used solely for the purposes of explaining and illustrating the concepts and procedures specified herein. There is no requirement that implementations represent WFNs explicitly or use this notation in any way.

5.3 Well-Formedness Criteria

WFNs MUST satisfy these criteria:

1. The attributes defined in Section 5.4 are the only permitted attributes in an attribute-value pair of a WFN.
2. Each permitted attribute may be used at most once. If an attribute is not used in a WFN, it is said to be *unspecified*, and its value defaults to the logical value ANY (cf. 5.5.1).
3. Attribute values of WFNs must satisfy the requirements specified in Section 5.5.

5.4 Attributes

The following attributes SHALL be used to form attribute-value pairs in WFNs:

1. part
2. vendor
3. product
4. version
5. update
6. edition
7. language
8. sw_edition
9. target_sw
10. target_hw
11. other

The edition attribute SHALL be considered *deprecated* in this specification and its use is discouraged except for backward compatibility with [CPE22]. This attribute will be referred to as the "legacy *edition*" attribute.

The attributes *sw_edition*, *target_sw*, *target_hw*, and *other* are newly introduced in this specification and are referred to collectively as the *extended attributes*.

5.5 Requirements on Attribute Values in WFNs

Attributes of WFNs SHALL be assigned one of the following values:

1. A logical value specified in Section 5.5.1;
2. A character string satisfying both (a) the requirements on string values specified in Section 5.5.2, <u>and</u> (b) the per-attribute value restrictions specified in Section 0.

5.5.1 Logical values of WFNs

An attribute of a general WFN may be assigned one of these two logical values:

1. ANY (i.e., "any value");
2. NA (i.e., "not applicable/no value");

The logical value ANY should be assigned to an attribute when the creator of the WFN intends to express the idea that there are no restrictions on acceptable values for that attribute of the product being described or identified. The logical value NA should be assigned when the creator intends to express the idea that there is no legal or meaningful value for that attribute of the product being described or identified. In this specification we treat these two situations as equivalent: the situation in which an attribute is known to have no legal or meaningful value, and the situation in which the attribute has an obtainable value which is null. In both situations the logical value NA should be used.

At the Naming level of the CPE specification stack these distinctions have no special interpretation except that different binding rules may apply. At higher levels of the stack, however, these distinctions may be given special interpretations which impact behavior.

When transcribing WFNs in which these logical values appear, the values will be written in all uppercase characters, without surrounding quotation marks, on the right side of the equal sign, as in the examples below:

- `wfn:[…,update=ANY,…]`
- `wfn:[…,update=NA,…]`

5.5.2 Restrictions on attribute value strings

Value strings assigned to attributes of WFNs SHALL be *non-empty contiguous strings of bytes* encoded using the American Standard Code for Information Interchange (US-ASCII, also known as ANSI_X3.4-1968).

When transcribing value strings in WFNs, they will be enclosed in double quotes as in the examples below. The quotation marks are, of course, not considered part of the string values themselves.

- `wfn:[…,update="sr1",…]`
- `wfn:[…,target_hw="x64",update="sp2"]`

Value strings in WFNs SHALL satisfy all of the following general requirements:

a. Any lowercase letter or digit character may be used (ASCII decimals 48-57 and 97-122).

b. The *underscore* (decimal 95) may be used, and is recommended for use in place of whitespace characters (which are not permitted).
c. The backslash (decimal 92) is designated the *escape character*. It should be used in a value string when required to modify the interpretation of the character that immediately follows (see below). In these circumstances, the character following the backslash is said to be *quoted*.
d. The *asterisk* (decimal 42) and the *question-mark* (decimal 63) are designated *special characters*. These two characters may be assigned special interpretations at higher levels of the CPE specification stack. To block special interpretation of these characters, precede them with the escape character, otherwise, leave them unquoted in the value string.
e. All other *printable non-alphanumeric characters* (i.e., all punctuation marks, brackets, delimiters and other special-purpose symbols, except for the special characters defined above) must be quoted when embedded in attribute value strings of WFNs.

These requirements are summarized by the ABNF grammar for *avstring* shown below in Figure 5-1.

```
avstring      = +(unreserved / special / quoted)
unreserved    = LCALPHA / DIGIT / "_"
quoted        = escape (escape / special / punc)
escape        = "\"
special       = "?" / "*"
punc          = "." / "-" / ":" / "/" / "#" / "[" / "]" / "@"
                / "~" / "!" / "$" / "&" / "'" / "(" / ")" / "+"
                / "," / ";" / "=" / "{" / "}" / "|" / "`" / "%"
                / "<" / ">" / "^" / DQUOTE
DQUOTE        = %x22 ; double quote
LCALPHA       = %x61-7A
DIGIT         = %x30-39
```

Figure 5-1: ABNF Grammar for Attribute Value Strings

Examples of allowable value strings in WFNs:

- `"foo\-bar"` (hyphen is quoted)
- `"acrobat_reader"`
- `"\"oh_my\!\""` (quotation marks and exclamation point are quoted)
- `"g\+\+"` (plus signs are quoted)
- `"9\.?"` (period is quoted, question-mark is unquoted)
- `"sr*"` (asterisk is unquoted)
- `"big\$money"` (dollar sign is quoted)
- `"foo\:bar"` (colon is quoted)
- `"back\\slash_software"` (backslash is quoted)

5.5.3 Per-attribute value restrictions

This section specifies value restrictions that may apply to specific attributes in a WFN. In addition, recommendations are provided for how suitable attribute value strings should be chosen.

5.5.3.1 Part

The *part* attribute SHALL be one of these three string values: “a”, “o”, and “h”.

The value “a” SHALL be used when the WFN is intended to describe or identify a class of *applications*.

The value “o” SHALL be used when the WFN is intended to describe or identify a class of *operating systems.*

The value “h” SHALL be used when the WFN is intended to describe or identify a class of *hardware devices.*

5.5.3.2 Vendor

For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) MAY be specified as the value of the *vendor* attribute. Values for this attribute SHOULD be selected from an attribute-specific valid-values list. Values for this attribute SHOULD describe or identify the person or organization that manufactured or created the product which is being described or identified by the WFN.

5.5.3.3 Product

For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) MAY be specified as the value of the *product* attribute. Values for this attribute SHOULD be selected from an attribute-specific valid-values list. Values for this attribute SHOULD describe or identify the most common and recognizable title or name of the product which is being described or identified by the WFN.

5.5.3.4 Version

For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) may be specified as the value of the *version* attribute. Values for this attribute SHOULD be vendor-specific alphanumeric strings characterizing the particular release version of the product which is being described or identified by the WFN. Version information SHOULD be copied directly (with escaping of printable non-alphanumeric characters as required) from discoverable data and not truncated or otherwise modified.

5.5.3.5 Update

For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) MAY be specified as the value of the *update* attribute. Values for this attribute SHOULD be selected from an attribute-specific valid-values list. Values for this attribute SHOULD be vendor-specific alphanumeric strings characterizing the particular update, service pack, or point release of the product which is being described or identified by the WFN.

5.5.3.6 Edition

In this Naming Specification, the *edition* attribute SHALL be considered deprecated, and its use is discouraged except where required for backward compatibility with version 2.2 of the CPE specification. This attribute is referred to as the "legacy *edition*" attribute.

For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) MAY be specified as the value of the legacy *edition* attribute. Values for this attribute SHOULD be selected from an attribute-specific valid-values list. Values for this attribute SHOULD capture edition-related terms applied by the vendor to the product which is being described or identified by the WFN.

5.5.3.7 SW_Edition

The *sw_edition* attribute is considered to be a member of the set of *extended attributes*. For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) MAY be specified as the value of the *sw_edition* attribute. Values for this attribute SHOULD be selected from an attribute-specific valid-values list. Terms used for this attribute SHOULD characterize how the product being described or identified by the WFN is tailored to a particular market or class of end users.

5.5.3.8 Target_SW

The *target_sw* attribute is considered to be a member of the set of *extended attributes*. For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) MAY be specified as the value of the *target_sw* attribute. Values for this attribute SHOULD be selected from an attribute-specific valid-values list. Terms used for this attribute SHOULD characterize the software computing environment within which the product being described or identified by the WFN operates.

5.5.3.9 Target_HW

The *target_hw* attribute is considered to be a member of the set of *extended attributes*. For the purposes of this Naming specification, any character string meeting the requirements for WFNs (cf. Section 5.5.2) MAY be specified as the value of the *target_hw* attribute. Values for this attribute SHOULD be selected from an attribute-specific valid-values list. Terms used for this attribute SHOULD characterize the physical computing platform on which the product being described or identified by the WFN operates.

5.5.3.10 Language

The value of the *language* attribute SHALL be a valid language tag as defined by [RFC4646]. Although any valid language tag is acceptable, WFNs SHOULD only use tags containing language and region codes.

5.5.3.11 Other

The value of the *other* attribute SHOULD be used to capture any other general descriptive or identifying information which is vendor- or product-specific and which does not logically fit in any other attribute value of the WFN. Values for this attribute SHOULD be selected from a valid-values list that is refined over time.

5.6 Operations on WFNs

Three functions are defined over WFNs: *new*, *get*, and *set*. These functions will be useful when defining binding and unbinding procedures in Section 6.

5.6.1 Function new()

The `new()` function takes no arguments. The `new()` function returns an *empty WFN* (a WFN containing no attribute-value pairs).

Example:

```
new() → wfn:[]
```

5.6.2 Function get(w,a)

The `get(w,a)` accessor function takes two arguments, a WFN *w* and an attribute *a*, and returns the value of *a*. If the attribute *a* is unspecified in *w*, `get(w,a)` returns the default value ANY.

Examples:

- `get(wfn:[vendor="microsoft",product="internet_explorer"],vendor)`
  `→ "microsoft"`
- `get(wfn:[vendor="microsoft",product="internet_explorer"],version)`
  `→ ANY`

5.6.3 Function set(w,a,v)

The `set(w,a,v)` function takes three arguments, a WFN *w*, an attribute *a*, and a value *v*. If the attribute *a* is unspecified in *w*, `set(w,a,v)` adds the attribute-value pair *a*=*v* to *w*. If the attribute *a* is specified in *w*, `set(w,a,v)` replaces its value with *v* in *w*. If *v* is *nil*, `set(w,a,v)` *deletes a* from *w* if *a* is specified in *w*, otherwise has no effect. The function always returns the new value of *w*.

Examples:

- `set(wfn:[], vendor, "microsoft") → wfn:[vendor="microsoft"]`
- `set(wfn:[vendor="microsoft"], vendor, "adobe") →`
  `wfn:[vendor="adobe"]`
- `set(wfn:[vendor="microsoft"], update, ANY) →`
  `wfn:[vendor="microsoft",update=ANY]`
- `set(wfn:[vendor="microsoft"], vendor, nil) = wfn:[]`

5.7 Examples

This section illustrates a variety of WFNs. The examples below are intended only to illustrate names that are well formed according to the rules defined above. These examples do not necessarily illustrate "correct" or "valid" assignments of values to attributes.

- Microsoft Internet Explorer 8.0.6001 Beta (no edition):
  `wfn:[part="a",vendor="microsoft",product="internet_explorer",`
  `version="8\.0\.6001",update="beta",edition=NA]`

- Microsoft Internet Explorer 8.* SP? (no edition, any language):
```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update= "sp? ",edition=NA,language=ANY]
```

- Identifier for HP Insight Diagnostics 7.4.0.1570 Online Edition for Windows 2003 x64:
```
wfn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.0\.1570",sw_edition="online",
target_sw="windows_2003",target_hw="x64"]
```

- Identifier for HP OpenView Network Manager 7.51 (no update) for Linux:
```
wfn:[part="a",vendor="hp",product="openview_network_manager",
version="7\.51",update=NA,target_sw="linux"]
```

- Foo\Bar Systems Big$Money 2010 Special Edition for iPod Touch 80GB:
```
idn:[part="a",vendor="foo\\bar",product="big\$money_2010",
sw_edition="special",target_sw="ipod_touch",target_hw="80gb"]
```

6. Implementation and Binding

This section defines the procedures for *binding* (cf. 2.1.3) WFNs to machine-readable representations, as well as the procedures for *unbinding* (cf. 2.1.17) machine-readable representations into WFNs.

6.1 Notes on Pseudo-Code

This document uses an abstract pseudo-code programming language to specify expected computational behavior. Pseudo-code is intended to be straightforwardly readable and translatable into actual programming language statements. Note, however, that pseudo-code specifications are not necessarily intended to illustrate efficient or optimized programming code; rather, their purpose is to clearly define the desired behavior, leaving it to implementers to choose the best language-specific design which respects that behavior. In some cases, particularly where standardized implementations exist for a given pseudo-code function, we describe the function's behavior in prose.

In reading pseudo-code the following notes should be kept in mind:

- All pseudo-code functions are *pass by reference*, meaning that any changes applied to the supplied arguments within the scope of the scope of the function do not affect the values of the variables in the caller's scope.
- In a few cases, the pseudo-code functions reference (more or less) standard library functions, particularly to support string handling. In most cases semantically equivalent functions can be found in the GNU C library, cf. http://www.gnu.org/software/libc/manual/html_node/index.html#toc_String-and-Array-Utilities.

6.2 URI Binding

The URI Binding is included here for backward compatibility with prior releases of the CPE specification. Section 5.1 of [CPE22] specifies that a CPE name is a percent-encoded URI [RFC3986] with each name having the URI scheme name "cpe:". The procedure defined here for creating a URI binding ensures that when a WFN is bound to a URI, it will satisfy the requirements of [CPE22] for CPE names.

Section 6.2.1 defines the syntax of a valid URI binding. Section 0 specifies the procedure for binding a WFN to a URI. Section 6.2.3 specifies the procedure for unbinding a URI into a WFN. It is important to note that the binding and unbinding functions on URIs are not necessarily *symmetric*—that is, if one binds a WFN *w1* to a URI, and then unbinds the result to a WFN *w2*, it is not guaranteed that *w1 = w2*. This is due to the fact that certain WFN capabilities introduced in this specification document did not exist in [CPE22] and thus cannot be encoded in a v2.2-conformant URI. So meaning may be lost in the process of binding a given WFN to a URI, and this meaning cannot be recovered by the unbinding procedure.

6.2.1 URI Binding Syntax

The syntax of legal CPE URIs is specified in Appendix A of [CPE22]. It is included here in ABNF notation [RFC2234] for ease of reference.

```
cpe-name          = "cpe:/" component-list

component-list    = part ":" vendor ":" product ":" version ":" update ":"
                    edition ":" lang
component-list    /= part ":" vendor ":" product ":" version ":" update ":"
                     edition
component-list    /= part ":" vendor ":" product ":" version ":" update
component-list    /= part ":" vendor ":" product ":" version
component-list    /= part ":" vendor ":" product
component-list    /= part ":" vendor
component-list    /= part
component-list    /= empty

part              = "h" / "o" / "a" / empty
vendor            = string
product           = string
version           = string
update            = string
edition           = string
lang              = LANGTAG / empty
string            = *( unreserved / pct-encoded )
empty             = ""

unreserved        = ALPHA / DIGIT / "-" / "." / "_" / "~" / “%”
pct-encoded       = "%" HEXDIG HEXDIG
ALPHA             = %x41-5A / %x61-7A   ; A-Z / a-z
DIGIT             = %x30-39  ; 0-9
HEXDIG            = DIGIT / "a" / "b" / "c" / "d" / "e" / "f"
LANGTAG           = cf. [RFC4646]
```

Figure 6-1: ABNF for URI Binding

6.2.2 Binding a WFN to a URI

Given a WFN, the procedure to bind it to a URI is specified in pseudo-code below. The top-level binding function, `bind_to_URI`, is called with the WFN to be bound as its only argument. The pseudo-code references the defined operations on WFNs (cf. 5.6) as well as a number of helper functions also defined in pseudo-code. Section 6.2.2.1 provides some important notes on the binding procedure. Section 6.2.2.2 summarizes the algorithm in prose. Section 6.2.2.3 provides the pseudo-code for the algorithm. Section 6.2.2.4 provides examples of binding WFNs to URIs. The algorithm defined here assumes that the input WFN is well formed according to the well-formedness criteria defined in Section 5.3. The behavior of `bind_to_URI` is undefined if its input is not well formed.

6.2.2.1 Notes on URI binding procedure

The procedure for binding WFNs to URIs has three noteworthy properties.

1. **Handling of logical values:** In WFNs, two logical values (ANY and NA) are defined. The logical value ANY is bound to what [CPE22] calls a “blank” (i.e., a null character between two colons) in the URI. The logical value NA is bound to a single hyphen.
2. **Handling of non-alphanumeric characters:** In WFN attribute value strings, non-alphanumeric characters must be quoted, though the special characters “*” and “?” may appear without quoting. [CPE22] requires that most non-alphanumerics be percent encoded, and makes no allowance for

those characters to appear without percent encoding. So all quoting must be removed as part of the binding procedure, followed by percent encoding as required by [CPE22]. As a result, both quoted and unquoted special characters end up being percent encoded in the URI form—a second aspect in which the URI binding procedure is lossy.

3. **Packing:** This specification introduces four new attributes—the extended attributes—which have no assigned position in the URI binding. When these attributes have values other than ANY in the WFN, they are "packed" in a special format, and in a specified order, into the edition component of the URI. This special format uses the tilde character "~" as a sub-delimiter. Consequently, the binding procedure *deletes* any tilde characters if they are embedded in the value strings. This is a third aspect in which the URI binding procedure is lossy.

As noted above, the URI binding procedure is lossy in several ways. The capability to bind WFNs to URIs is provided primarily for use by dictionary creators and maintainers, to allow them to create new CPE names that take full advantage of all features introduced in this specification, while still having a backward-compatible path for creating approximate names that conform to [CPE22]. This capability should be used with care as CPE v2.2-conformant tools may be unable to properly match names that differ in terms of packed attribute values.

6.2.2.2 Summary of algorithm

The URI binding procedure is summarized as follows:

1. Initialize the output URI binding to the string "cpe:/".
2. BEGIN LOOP: Iterate over the seven attributes corresponding to the seven components in a v2.2 CPE URI [CPE22]. Get the value of each attribute and perform steps 3 thru 7.
3. SPECIAL HANDLING OF EDITION: When binding to a 2.2 URI, the edition component (the sixth element of the URI) is used as the location to "pack" five attribute values in the WFN: (legacy) edition, sw_edition, target_sw, target_hw, and other. The "packing" process involves concatenating the five values together, prefixed and separated by the tilde character (which is not allowed to be used in attribute value strings). The leading tilde serves as a flag indicating that the contents of the edition field are a packed representation of five separate values, and the internal tildes are used to aid parsing the values out. In the special case in which the four extended attributes are not specified, or all are ANY, only the edition attribute is used and no packing is performed.
4. BIND ATTRIBUTE VALUES:
    a. For all attributes *other than* (legacy) "edition", inspect the value and convert logical values appropriately. If the attribute is unspecified, or its logical value is ANY, bind it to blank ("") in the URI. If the logical value is NA, bind it to the hyphen ("-").
    b. REMOVE ESCAPING: Scan the attribute value for any escaped characters and simply remove the escaping.
    c. APPLY PERCENT-ENCODING: Percent-encode all reserved characters remaining in the attribute value string as required by [RFC3896].
5. Append the attribute value string to the output URI, followed by a trailing colon.
6. END LOOP.
7. Return the output URI, trimming away all trailing colons for compactness.

6.2.2.3 Pseudo-code for algorithm

```
function bind_to_URI(w)
  ;; Top-level function used to bind a WFN w to a URI.
  ;; Initialize the output with the CPE v2.2 URI prefix.
```

```
    uri := “cpe:/”.
    for each a in {part,vendor,product,version,update,edition,language}
      do
        if a = edition
          then
            ;; Call the pack() helper function to compute the proper
            ;; binding for the edition element.
            ed := bind_value_for_URI(get(w,edition)).
            sw_ed := bind_value_for_URI(get(w,sw_edition)).
            t_sw := bind_value_for_URI(get(w,target_sw)).
            t_hw := bind_value_for_URI(get(w,target_hw)).
            oth := bind_value_for_URI(get(w,other)).
            v := pack(ed,sw_ed,t_sw,t_hw,oth).
          else
            ;; Get the value for a in w, then bind to a string
            ;; for inclusion in the URI.
            v := bind_value_for_URI(get(w,a)).
        endif.
      ;; Append v to the URI then add a colon.
      uri := strcat(uri, v, “:”).
    end.
    ;; Return the URI string, with trailing colons trimmed.
    return trim(uri).
  end.

  function bind_value_for_URI(s)
    ;; Takes a string s and converts it to the proper string for
    ;; inclusion in a CPE v2.2-conformant URI.  The logical value ANY
    ;; binds to the blank in the 2.2-conformant URI.
    if s = ANY then return(“”).
    ;; The value NA binds to a single hyphen.
    if s = NA then return(“-”).
    ;; If we get here, we’re dealing with a string value.
    ;; In the URI, there is no quoting, so strip out any escape chars.
    s := delete_char(s,”\”).
    ;; Percent-encode non-alphanumerics as required by [CPE22].
    s := pct_encode(s).
    return s.
  end.

  function delete_char(s,badchar)
    ;; Returns a copy of string s with all instances of character
    ;; badchar removed.
    result := “”.
    idx := 0.
    while (idx < strlen(s)) do
      thischar := substr(s,idx,1).  ; get the idx’th character of s.
      if (thischar != badchar)
        then
          ;; copy this to result.
          result := strcat(result,thischar).
      endif.
```

```
    idx := idx + 1.
  end.
  return result.
end.

function pct_encode(s)
  ;; Return s with any reserved characters percent-encoded.
  ;; We leave the implementation unspecified as there are
  ;; standardized algorithms for percent encoding.  Only certain
  ;; characters embedded in s should be percent encoded as
  ;; follows:
  ;; '!' -> "%21" (exclamation mark)
  ;; '"' -> "%22" (double quote)
  ;; '#' -> "%23" (pound sign)
  ;; '$' -> "%24" (dollar sign)
  ;; '%” -> "%25" (percent sign)
  ;; '&' -> "%26" (ampersand)
  ;; '’' -> "%27" (apostrophe)
  ;; '(' -> "%28" (left paren)
  ;; ')' -> "%29" (right paren)
  ;; '*' -> "%2A" (asterisk)
  ;; '+' => "%2B" (plus sign)
  ;; ',' -> "%2C" (comma)
  ;; '/' -> "%2F" (forward slash)
  ;; ':' -> "%3A" (colon)
  ;; ';' -> "%3B" (semi-colon)
  ;; '<' -> "%3C" (left angle bracket)
  ;; '=' -> "%3D" (equal sign)
  ;; '>' -> "%3E" (right angle bracket)
  ;; '?' -> "%3F" (question mark)
  ;; '@' -> "%40" (at sign)
  ;; '[' -> "%5B" (left bracket)
  ;; ']' -> "%5D" (right bracket)
end.

function pack(ed,sw_ed,t_sw,t_hw,o)
  ;; “Pack” the values of the five arguments into the single edition
  ;; component.  If all the values are blank, just return a blank.
  if (sw_ed = "" and t_sw = "" and t_hw = "" and o = "")
    then
      ;; All the extended attributes are blank, so don’t do
      ;; any packing, just return ed.
      return ed.
  end.
  ;; Otherwise, pack the five values into a single string
  ;; prefixed and internally delimited with the tilde.
  ;; Because the tilde is used as a sub-delimiter, we must
  ;; delete it if it’s embedded in any of the value strings
  ;; to be packed.
  ed := delete_char(ed,'~').
  sw_ed := delete_char(sw_ed,'~').
  t_sw := delete_char(t_sw,'~').
```

```
  t_hw := delete_char(t_hw,'~').
  o := delete_char(o,'~').
  return strcat('~',ed,'~',sw_ed,'~',t_sw,'~',t_hw,'~',o).
end.

function trim(s)
  ;; Remove trailing colons from the URI back to the first non-colon.
  s1 := reverse(s).
  idx := 0.
  for i := 0 to strlen(s1) do
    if substr(s1,i,1) = ":"
      then idx := idx + 1.
      else break.
  end.
  ;; Return the substring after all trailing colons,
  ;; reversed back to its original character order.
  return(reverse(substr(s1,idx,strlen(s1)-1))).
end.

function strcat(s1, s2, ... sn)
  ;; Returns a copy of the string s1 with the strings s2 to sn
  ;; appended in the order given.
  ;; Cf. the GNU C definition of strcat.  This function shown
  ;; here differs only in that it can take a variable number
  ;; of arguments.  This is really just shorthand for,
  ;; strcat(s1, strcat(s2, strcat(s3, … ))).
end.

function strlen(s)
  ;; Defined as in GNU C, returns the length of string s.
  ;; Returns zero if the string is empty.
end.

function substr(s,b,e)
  ;; Returns a substring of s, beginning at the b’th character,
  ;; with 0 being the first character, and ending at the e’th
  ;; character.  B must be <= E.  Returns nil if b >= strlen(s).
end.

function reverse(s)
  ;; Returns a reverse copy of string S, i.e., the last character
  ;; becomes the first character, the second-to-last becomes the
  ;; second character, etc.
end.
```

6.2.2.4 Examples of binding a WFN to a URI

This section illustrates several examples of binding WFNs to URIs.

6.2.2.4.1 Example 1

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.0.6001 Beta (any language):

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY]
```

This WFN binds to the following URI:

```
cpe:/a:microsoft:internet_explorer:8.0.6001:beta
```

Note how the trailing colons are removed, such that the “edition=ANY” effectively disappears.

6.2.2.4.2 Example 2

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.* SP?:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update="sp? "]
```

This WFN binds to the following URI:

```
cpe:/a:microsoft:internet_explorer:8.%42:sp%63
```

Note how the unquoted special characters in the WFN get percent-encoded in the URI. Their special functionality in the WFN does not translate to a 2.2 URI, and any special meanings are lost. If the above binding were unbound (see Section 6.2.3), the asterisk and question mark would be *quoted* in the resulting WFN.

6.2.2.4.3 Example 3

Suppose one had created the WFN below to describe this product: HP Insight Diagnostics 7.4.0.1570 Online Edition for Windows 2003 x64:

```
idn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.0\.1570",update=NA,
sw_edition="online",target_sw="win2003",target_hw="x64"]
```

This WFN binds to the following URI:

```
cpe:/a:hp:insight_diagnostics:7.4.0.1570:-:~~online~win2003~x64~
```

Note how the legacy edition attribute as well as the four extended attributes are packed into the edition component of the URI.

6.2.2.4.4 Example 4

Suppose one had created the WFN below to describe this product: HP OpenView Network Manager 7.51 (any update) for Linux:

```
wfn:[part="a",vendor="hp",product="openview_network_manager",
version="7\.51",target_sw="linux"]
```

This WFN binds to the following URI:

```
cpe:/a:hp:openview_network_manager:7.51::~~~linux~~
```

Note how the unspecified update attribute binds to a blank in the URI, and how packing occurs in the edition component when only the target_sw attribute is specified.

6.2.2.4.5 Example 5

Suppose one had created the WFN below to describe this product: Foo\Bar Big$Money Manager 2010 Special Edition for iPod Touch 80GB:

```
wfn:[part="a",vendor="foo\\bar",product="big\$money_manager_2010",
sw_edition="special",target_sw="ipod_touch",target_hw="80gb"]
```

This WFN binds to the following URI:

```
cpe:/a:foo\bar:big%24money_manager_2010:::~~special~ipod_touch~80gb~
```

Note how the \\ becomes a single backslash that is not percent-encoded because it's allowed in a URI. Also note how the dollar sign is percent-encoded, and how the extended attributes are packed.

6.2.3 Unbinding a URI to a WFN

Given a CPE v2.2-conformant URI, the procedure to unbind it to a WFN is specified in pseudo-code below. The top-level unbinding function, `unbind_URI`, is called with the URI to be unbound as its only argument. The pseudo-code references the defined operations on WFNs (cf. 5.6) as well as a number of helper functions also defined in pseudo-code. Section 6.2.3.1 summarizes the algorithm in prose. Section 6.2.3.2 provides the pseudo-code for the algorithm. Section 6.2.3.3 provides examples of unbinding URIs to WFNs. Note that the pseudo-code below reuses a number of helper functions defined above in Section 6.2.2.3. The algorithm defined here assumes that the input URI conforms to the CPE v2.2 specification. (This is guaranteed if the URI is the result of binding a WFN.) The behavior of `unbind_URI` is undefined otherwise.

6.2.3.1 Summary of algorithm

The procedure for unbinding a URI is straightforward:

1. Loop over the seven attributes corresponding to the seven CPE v2.2 components, performing steps 2 through 7.
2. Parse out the string in the corresponding field of the URI.
3. Decode any characters which are percent encoded.
4. Insert the escape character preceding all non-alphanumerics.
5. Inspect the value and unbind it if necessary into the appropriate logical value. The lone hyphen unbinds to the logical value NA, and the blank unbinds to the logical value ANY.
6. Unpack the edition component if a leading tilde indicates it contains a packed collection of five attribute values.
7. Set the attribute value in the WFN to the determined value.

6.2.3.2 Pseudo-code for algorithm

```
function unbind_URI(uri)
  ;; Top-level function used to unbind a URI uri to a WFN.
  ;; Initialize the empty WFN.
  result := new().
  for i := 1 to 7
```

```
    do
      v := get_comp_uri(uri,i). ; get the i’th component of uri
      ;; unbind the parsed string.
      case v:
        '': v := ANY.  ; convert a blank to logical ANY.
        '-': v:= NA.   ; convert a hyphen to logical NA.
        else:
          v := pct_decode(v).
      end.
      case i:
        1: result := set(result,part,add_escaping(v)).
        2: result := set(result,vendor,add_escaping(v)).
        3: result := set(result,product,add_escaping(v)).
        4: result := set(result,version,add_escaping(v)).
        5: result := set(result,update,add_escaping(v)).
        6: ;; Special handling for edition component.
           ;; Unpack edition if needed.
           if (v = ANY or v = NA or substr(v,0,1) != "~")
             then
               ;; Just a logical value or a non-packed value.
               ;; So unbind to legacy edition, leaving other
               ;; extended attributes unspecified.
               result := set(result,edition,add_escaping(v)).
             else
               ;; We have five values packed together here
               result := unpack(v,result).
           end.
        7: result := set(result,language,add_escaping(v)).
      end.
    end.
    return result.
  end.

function unpack(s,wfn).
  ;; Argument s is a packed edition string, wfn is a WFN.
  ;; Unpack its elements and set the attributes in wfn accordingly.
  ;; Parse out the five elements.  This is an extremely crude
  ;; algorithm.
  start := 1.
  end := strchr(s,'~',start).
  if (start = end)
    then ed := "".
    else ed := substr(s,start,end-start).
  end.
  start := end+1.
  end := strchr(s,'~',start).
  if (start = end)
    then sw_ed := "".
    else sw_ed := substr(s,start,end-start).
  end.
  start := end+1.
  end := strchr(s,'~',start).
```

```
  if (start = end)
    then t_sw := "".
    else t_sw := substr(s,start,end-start).
  end.
  start := end+1.
  end := strchr(s,'~',start).
  if (start = end)
    then t_hw := "".
    else t_hw := substr(s,start,end-start).
  end.
  start := end+1.
  if (start >= strlen(s))
    then oth := "".
    else oth := substr(s,start,strlen(s)-start).
  end.
  wfn := set(wfn,edition,add_escaping(ed)).
  wfn := set(wfn,sw_edition,add_escaping(sw_ed)).
  wfn := set(wfn,target_sw,add_escaping(t_sw)).
  wfn := set(wfn,target_hw,add_escaping(t_hw)).
  wfn := set(wfn,other,add_escaping(oth)).
  return wfn.
end.

function add_escaping(s).
  ;; Scan the string s, looking for occurrences of printable
  ;; non-alphanumerics.  If found, add these to the output string
  ;; preceded by the escape character.
  result := "".
  idx := 0.
  while (idx < strlen(s))
    do
      c := substr(s,idx,1).  ; get the idx’th character of s.
      if (is_alphanum(c))
        then
          result := strcat(result,c).
        else
          result := strcat(result,'\’,c).
      end.
    idx := idx + 1.
  end.
  return result.
end.

function is_alphanum(c)
  ;; Returns TRUE iff c is an uppercase letter, a lowercase letter,
  ;; a digit, or the underscore, otherwise FALSE.
end.

function get_comp_uri(uri,i)
  ;; Return the i’th CPE component of the URI.  If i=0,
  ;; return the URI scheme.  For example, given URI:
  ;; cpe:/a:foo::bar
```

```
  ;; get_comp_uri(uri,0) = "cpe:"
  ;; get_comp_uri(uri,1) = "a"
  ;; get_comp_uri(uri,2) = "foo"
  ;; get_comp_uri(uri,3) = ""
  ;; get_comp_uri(uri,4) = "bar"
  ;; get_comp_uri(uri,5) = ""
  ;; etc.
end.

function pct_decode(s)
  ;; This function scans the string s and returns a copy
  ;; with all percent-encoded characters decoded.  This
  ;; function is the inverse of pct_encode(s) defined in
  ;; Section 6.2.2.3.  This function should be robust to
  ;; the possibility that ANY character, not just the required
  ;; printable non-alphanumeric characters, might be percent
  ;; encoded and will need to be properly decoded.
end.

function strchr(str,chr,off)
  ;; Searches the string str for the character chr starting
  ;; at offset off into the string.  Returns the offset of
  ;; the chr if found, otherwise nil.
  ;; Defined similar to the standard C function strchr.
  ;; But this version takes a third argument off, which
  ;; is an offset into the str to begin the search.
end.
```

6.2.3.3 Examples of unbinding a URI to a WFN

This section provides a number of examples illustrating the results of unbinding a URI to a WFN.

6.2.3.3.1 Example 1

URI: `cpe:/a:microsoft:internet_explorer:8.0.6001:beta`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY,
language=ANY]
```

Notice how legacy edition and all the extended attributes are unbound to the logical value ANY.

6.2.3.3.2 Example 2

URI: `cpe:/a:microsoft:internet_explorer:8.%42:sp%63`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.\*",update="sp\?",edition=ANY,language=ANY]
```

Note how the two percent-encoded special characters are unbound with added quoting.

6.2.3.3.3 Example 3

URI: `cpe:/a:hp:insight_diagnostics:7.4.0.1570::~~online~win2003~x64~`

Unbinds to this WFN:

```
wfn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.0\.1570",update=ANY,edition=ANY,
sw_edition="online",target_sw="win2003",target_hw="x64",
other=ANY]
```

Note how the legacy edition attribute as well as the four extended attributes are unpacked from the edition component of the URI.

6.2.3.3.4 Example 4

URI: `cpe:/a:hp:openview_network_manager:7.51:-:~~~linux~~`

Unbinds to this WFN:

```
wfn:[part="a",vendor="hp",product="openview_network_manager",
version="7\.51",update=NA,edition=ANY,sw_edition=ANY,
target_sw="linux",target_HW=ANY,other=ANY]
```

Note how the lone hyphen in the update component is unbound to the logical value NA, and how all the other blanks embedded in the packed edition component unbind to ANY, with only the target_sw attribute actually specified.

6.2.3.3.5 Example 5

URI: `cpe:/a:foo\bar:big%24money_2010:::~~special~ipod_touch~80gb~`

Unbinds to this WFN:

```
wfn:[part="a",vendor="foo\\bar",product="big\$money_2010",
version=ANY,update=ANY,edition=ANY,
sw_edition="special",target_sw="ipod_touch",target_hw="80gb",
other=ANY]
```

6.3 Formatted String Binding

The formatted string binding is new to v2.3 of the CPE specification suite. In keeping with the spirit of the v2.2 specification, the formatted string binding looks similar to the URI binding; however, it is defines simply to be a “formatted string” rather than a URI in order to relax the requirements that typically apply to URIs as specified in [RFC3986].

The formatted string binding is a colon-delimited list of fields prefixed with the string “cpe23:”. Use of a prefix distinct from the v2.2 URI binding enables tools to inspect a given input string and use a simple syntactic test to determine whether to process the input as a URI or as a formatted string. The formal syntax of the formatted string binding is presented in ABNF in Section 6.3.1.

Similar to the URI binding, the formatted string binds the attributes in a WFN in a fixed order, separated by the colon character:

```
cpe23: part : vendor : product : version : update : edition :
       language : sw_edition : target_sw : target_hw : other
```

In a formatted string binding, the alphanumeric characters plus hyphen (“-”), period (“.”) and underscore (“_”) appear explicitly. When used alone, the asterisk (“*”) represents the logical value ANY, and the hyphen (“-”) represents the logical value NA. All other non-alphanumeric characters, if used, must be preceded by the backslash. The special characters asterisk and question-mark may appear without a preceding backslash, in which case they are open to special interpretation at higher levels of the CPE specification stack.

6.3.1 Syntax for Formatted String Binding

The syntax of the formatted string binding is shown below.

```
formstring        = "cpe23:" component-list

component-list    = part ":" vendor ":" product ":" version ":" update ":"
                    edition ":" lang ":" sw_edition ":" target_sw ":"
                    target_hw ":" other

part              = "h" / "o" / "a" / logical
vendor            = avstring
product           = avstring
version           = avstring
update            = avstring
edition           = avstring
lang              = LANGTAG / logical
sw_edition        = avstring
target_sw         = avstring
target_hw         = avstring
other             = avstring

avstring          = +( unreserved / special / quoted ) / logical
logical           = "*" / "-"
special           = "*" / "?"
unreserved        = LCALPHA / DIGIT / "-" / "." / "_"
quoted            = escape (escape / special / punc)
escape            = "\"
punc              = "`" / "~" / "!" / "@" / "#" / "$" / "%" / "^" / "&"
                    / "(" / ")" / "=" / "+" / "[" / "{" / "]" / "}"
                    / "|" / ";" / ":" / "'" / DQUOTE / "<" / ">" / ","
                    / "/"
LCALPHA           = %x61-7A   ; a-z
DIGIT             = %x30-39  ; 0-9
DQUOTE            = %x22  ; double-quote
LANGTAG           = cf. [RFC4646]
```

Figure 6-2: ABNF for Formatted String Binding

6.3.2 Binding a WFN to a formatted string

This section specifies the procedure for binding a WFN to a formatted string. Section 6.3.2.1 summarizes the algorithm in prose. Section 6.3.2.2 presents the pseudo-code for the algorithm. Section 6.3.2.3 presents examples illustrating the results of binding various WFNs to formatted strings.

6.3.2.1 Summary of algorithm

The binding algorithm is very simple. The procedure iterates over the eleven (11) allowed attributes in a fixed order. Corresponding attribute values are obtained from the input WFN and conversions of logical values are applied. A result string is formed by concatenating the attribute values separated by colons.

6.3.2.2 Pseudo-code for algorithm

```
function bind_to_fs(w)
  ;; Top-level function used to bind WFN w to formatted string.
  ;; Initialize the output with the CPE v2.3 string prefix.
  fs := "cpe23:".
  for each a in {part,vendor,product,version,update,edition,language,
                 sw_edition,target_sw,target_hw,other}
    do
      v := bind_value_for_fs(get(w,a)).
      fs := strcat(fs,v,":").
  end.
  return trim(fs).
end.

function bind_value_for_fs(v)
  ;; Convert the value v to its proper string representation for
  ;; insertion into the formatted string.
  case v:
    ANY: return("*").
    NA: return("-").
    else: return process_escaped_chars(v).
  end.
end.

function process_escaped_chars(s)
  ;; Inspect each character in string s.  Certain nonalpha
  ;; characters pass thru without escaping into the result,
  ;; but most retain escaping.
  result := "".
  idx := 0.
  while (idx < strlen(s))
    do
      c := substr(s,idx,1).  ; get the idx’th character of s.
      if c != “\”
        then
          ;; un-escaped characters pass thru unharmed
          result := strcat(result,c).
        else
```

```
          ;; Escaped characters are examined
          nextchr := substr(s,idx+1,1).
          case nextchr:
            ;; the period, hyphen and underscore pass unharmed.
            ".":
            "-":
            "_": result := strcat(result,nextchr).
            else:
              ;; all others retain escaping
              result := strcat(result,"\",c).
              idx := idx + 2.
              continue.
          end.
      endif.
      idx := idx + 1.
  end.
  return result.
end.
```

6.3.2.3 Examples of binding a WFN to a formatted string

This section presents examples illustrating the results of binding various WFNs to formatted strings.

6.3.2.3.1 Example 1

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.0.6001 Beta (any language):

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY]
```

This WFN binds to the following formatted string:

```
cpe23:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*
```

Note how the unspecified attributes bind to "*" in the formatted string binding.

6.3.2.3.2 Example 2

Suppose one had created the WFN below to describe this product: Microsoft Internet Explorer 8.* SP? (any edition):

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update="sp?",edition=ANY]
```

This WFN binds to the following formatted string:

```
cpe23:a:microsoft:internet_explorer:8.*:sp?:*:*:*:*:*:*
```

Note how the unspecified attributes default to ANY and are thus bound to "*". Also note how the unquoted special characters in the WFN are carried over into the formatted string. Their special functionality in the WFN is preserved in the binding. If instead one wanted to block the special interpretation of the asterisk, it should be preceded by the escape character in the WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.\*",update="sp?"]
```

This WFN binds to the following formatted string:

```
cpe23:a:microsoft:internet_explorer:8.\*:sp?:*:*:*:*:*:*
```

In this case, the escape character appears explicitly in the binding, blocking the interpretation of the asterisk. The unquoted question mark retains any special interpretation it may have in the binding.

6.3.2.3.3 Example 3

Suppose one had created the WFN below to describe this product: HP Insight Diagnostics 7.4.0.1570 Online Edition for Windows 2003 x64:

```
wfn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.1570",update=NA,
sw_edition="online",target_sw="win2003",target_hw="x64"]
```

This WFN binds to the following formatted string:

```
cpe23:a:hp:insight_diagnostics:7.4.1570:-:*:*:online:win2003:x64:*
```

Notice how the NA binds to the lone hyphen, the unspecified edition, language and other all bind to the asterisk, and the extended attributes appear in their own fields.

6.3.2.3.4 Example 4

Suppose one had created the WFN below to describe this product: HP OpenView Network Manager 7.51 (any update) for Linux:

```
wfn:[part="a",vendor="hp",product="openview_network_manager",
version="7\.51",target_sw="linux"]
```

This WFN binds to the following formatted string:

```
cpe23:a:hp:openview_network_manager:7.51:*:*:*:*:linux:*:*
```

Note how the unspecified attributes update, edition, language, sw_edition, target_hw, and other all bind to an asterisk in the formatted string.

6.3.2.3.5 Example 5

Suppose one had created the WFN below to describe this product: Foo\Bar Big$Money 2010 Special Edition for iPod Touch 80GB:

```
wfn:[part="a",vendor="foo\\bar",product="big\$money_2010",
sw_edition="special",target_sw="ipod_touch",target_hw="80gb"]
```

This WFN binds to the following formatted string:

```
cpe23:a:foo\\bar:big\$money_2010:*:*:*:*:special:ipod_touch:80gb:*
```

Note how the \\ and \$ carry over into the binding, and how all the other unspecified attributes bind to the asterisk.

6.3.3 Unbinding a formatted string to a WFN

Given a formatted string binding, the procedure to unbind it to a WFN is specified in pseudo-code below. The top-level unbinding function, `unbind_fs`, is called with the formatted string to be unbound as its only argument. The pseudo-code references the defined operations on WFNs (cf. 5.6) as well as a number of helper functions also defined in pseudo-code. Section 6.3.3.1 summarizes the algorithm in prose. Section 6.3.3.2 provides the pseudo-code for the algorithm. Section 6.3.3.3 provides examples of unbinding formatted strings to WFNs.

6.3.3.1 Summary of algorithm

Unbinding a formatted string is very simple, since the attribute values are encoded explicitly and in a fixed left-to-right order in the binding, delimited by colons. (Because a colon may appear embedded in a value string if preceded by the escape character, the parsing function needs to ignore escaped colons.) The algorithm parses the eleven fields of the formatted string, then unbinds each string result. If a field contains only an asterisk, it is unbound to the logical value ANY. If a field contains only a hyphen, it is unbound to the logical value NA. Quoting of non-alphanumeric characters is restored as needed, but the two special characters (asterisk and question-mark) are permitted to appear without a preceding escape character.

6.3.3.2 Pseudo-code for algorithm

```
function unbind_fs(fs)
  ;; Top-level function to unbind a formatted string fs to a wfn.
  result := new().
  for a = 1 to 11
    do
      v := get_comp_fs(fs,a).    ; get the a'th field string
      v := unbind_value_fs(v).   ; unbind the string
      ;; set the value of the corresponding attribute.
      case a:
        1: result := set(result,part,v).
        2: result := set(result,vendor,v).
        3: result := set(result,product,v).
        4: result := set(result,version,v).
        5: result := set(result,update,v).
        6: result := set(result,edition,v).
        7: result := set(result,language,v).
        8: result := set(result,sw_edition,v).
        9: result := set(result,target_sw,v).
       10: result := set(result,target_hw,v).
       11: result := set(result,other,v).
      end.
  end.
  return result.
end.

function get_comp_fs(fs,i)
  ;; Return the i'th field of the formatted string.  If i=0,
  ;; return the string to the left of the first forward slash.
```

```
  ;; The colon is the field delimiter unless prefixed by a
  ;; backslash.
  ;; For example, given the formatted string:
  ;; cpe23:a:foo:bar\:mumble:1.0:*:…
  ;; get_comp_fs(fs,0) = "cpe23"
  ;; get_comp_fs(fs,1) = "a"
  ;; get_comp_fs(fs,2) = "foo"
  ;; get_comp_fs(fs,3) = "bar\:mumble"
  ;; get_comp_fs(fs,4) = "1.0"
  ;; etc.
end.

function unbind_value_fs(s)
  ;; Takes a string value s and returns the appropriate logical
  ;; value if s is the bound form of a logical value.  If s is some
  ;; general value string, add escaping of non-alphanumerics as
  ;; needed.
  case s:
    "*": return ANY.
    "-": return NA.
    else:
      ;; add escaping to any unquoted non-alphanumeric characters,
      ;; but leave the two special characters alone, as they may
      ;; appear quoted or unquoted.
      return add_escaping(s).
  end.
end.

function add_escaping(s)
  ;; Inspect each character in string s.  Copy quoted characters,
  ;; with their escaping, into the result.  Look for unquoted non
  ;; alphanumerics and if not “*” or “?”, add escaping.
  result := "".
  idx := 0.
  while (idx < strlen(s))
    do
      c := substr(s,idx,1).  ; get the idx’th character of s.
      if (is_alphanum(c) or c = "*" or c = "?") then
        ;; letters, digits, underscores pass untouched,
        ;; and the same goes for the two special characters.
        result := strcat(result,c).
        idx := idx + 1.
        continue.
      endif.
      if c = "\" then
        ;; anything escaped in the bound string stays escaped
        ;; in the unbound string.
        result := strcat(result,substr(s,idx,2)).
        idx := idx + 2.
        continue.
      endif.
      ;; all other characters must be escaped
```

```
        result := strcat(result, "\",c).
        idx := idx + 1.
    end.
    return result.
end.
```

6.3.3.3 Examples of unbinding a formatted string to a WFN

This section provides a number of examples illustrating the results of unbinding a formatted string to a WFN.

6.3.3.3.1 Example 1

FS: `cpe23:a:microsoft:internet_explorer:8.0.6001:beta:*:*:*:*:*:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.0\.6001",update="beta",edition=ANY,language=ANY,
sw_edition=ANY,target_sw=ANY,target_hw=ANY,other=ANY]
```

Notice how the periods in the version string are quoted in the WFN, and all the asterisks are unbound to the logical value ANY.

6.3.3.3.2 Example 2

FS: `cpe23:a:microsoft:internet_explorer:8.*:sp?:*:*:*:*:*:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="microsoft",product="internet_explorer",
version="8\.*",update="sp?",edition=ANY,language=ANY,
sw_edition=ANY,target_sw=ANY,target_hw=ANY,other=ANY]
```

Note how the embedded special characters are unbound untouched in the WFN.

6.3.3.3.3 Example 3

FS: `cpe23:a:hp:insight_diagnostics:7.4.1570:-:*:*:online:win2003:x64:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="hp",product="insight_diagnostics",
version="7\.4\.0\.1570",update=NA,edition=ANY,language=ANY,
sw_edition="online",target_sw="win2003",target_hw="x64",
other=ANY]
```

Note how the lone hyphen in the update field unbinds to the logical value NA, and how the lone asterisks unbind to the logical value ANY.

6.3.3.3.4 Example 4

FS: `cpe23:a:foo\\bar:big\$money:2010:*:*:*:special:ipod_touch:80gb:*`

Unbinds to this WFN:

```
wfn:[part="a",vendor="foo\\bar",product="big\$money",
version="2010",update=ANY,edition=ANY,language=ANY,
sw_edition="special",target_sw="ipod_touch",target_hw="80gb",
other=ANY]
```

Note how the quoted special characters retain their quoting in the WFN.

7. Conversions

This section specifies the procedures for converting between the two required bound forms of WFNs. Section 7.1 specifies the procedure for converting a URI binding to a formatted string binding, and Section 7.2 specifies the inverse conversion.

7.1 Converting a URI to a Formatted String

Given a URI *u* which conforms to the CPE v2.2 specification, the procedure for converting it to a formatted string *fs* has two steps:

```
function convert_uri_to_fs(u)
  w := unbind_uri(u).
  fs := bind_to_fs(w).
  return fs.
end.
```

Note:

If one starts with a URI (e.g., a legacy CPE name from the v2.2 official dictionary), converts it to a formatted string, then back to a URI (using `convert_fs_to_uri` in Section 7.2), one will end up with the same URI one started with. That is, the URI-FS-URI conversion path is *round trip safe*.

7.2 Converting a Formatted String to a URI

Given a formatted string *fs* which conforms to the description in Section 6.3.2, the procedure for converting it to a URI has two steps:

```
function convert_fs_to_uri(fs)
  w := unbind_fs(fs).
  uri := bind_to_uri(w).
  return uri.
end.
```

Notes:

Note that if one starts with a formatted string, converts it to a URI, then back to a formatted string (using `convert_uri_to_fs` in Section 7.1), there is no guarantee that one will end up with the same formatted string one started with. The formatted string binding allows the introduction of new features that are unsupported in the backward-compatible URI binding; these features, if used, will not survive a round-trip conversion process. That is, the FS-URI-FS conversion path is <u>not</u> *round trip safe*. The conversion to a backward-compatible name form is specified here principally for use by curators of v2.3-conformant dictionaries, so they can automatically convert newly-created names into a backward-compatible format for use by legacy tools. Such cross-version interoperability cannot be fully supported, however, given the new features of the v2.3 CPE Specification Stack.

Appendix A—Use Cases

There are many areas within the security automation community which can benefit from CPE. Over the course of CPE's development, four use cases emerged as primary drivers of technical requirements:

1. Software inventory
2. Network-based discovery
3. Forensic analysis/system architecture
4. IT management

We summarize these use cases in the next four subsections.[6] Note that version 2.2 of the CPE specification was intended primarily to support the Software Inventory Use Case. The new version of CPE specified using the CPE Specification Stack is still primarily focused on the Software Inventory Use Case, however, by adding support for wildcards (using the special characters asterisk and question mark) we have attempted to expand the scope of CPE somewhat to include network-based discovery.

A.1 Software Inventory Use Case

Software inventory management products include configuration audit, endpoint management, and asset inventory tools. Such products typically have credentialed (authenticated) access to end systems. In this technical use case, a software inventory management product vendor uses CPE Names to tag data elements within their product's data model. These data elements may directly represent the individual software products that exist on a computing endpoint (e.g., a laptop, desktop, or server), in which case the CPE Name represents a standardized identifier for instances of that record type. Alternatively, the data elements may represent some other object (e.g., a configuration check, a vulnerability check, a patch check, a configuration control change, or a patch), in which case the CPE Name implies a relationship to a software product as identified by the CPE Name. With this tagging, the product vendor can develop, or can enable their product to interoperate with, different tools that share information about the individual software products on the end systems. Whether those tools perform asset management, vulnerability management, configuration assessments, or tactical descriptions of a given network, they have a common need to share software inventory information. The tools are expected to use CPE Names for this purpose.

A.2 Network-Based Discovery Use Case

Some enterprise users and tool developers are involved with network-based discovery of information that is performed without credentialed access to end systems. Their desire is to use CPE to tag the assets found and thus enable sharing of information with other information data sources. Unfortunately, unauthenticated network-based discovery often results in only partial information. Sometimes, full details cannot be determined in this way, but can only be obtained by a credentialed access to the end system. This results for the need for terms like "linux" or "printer" when the discovery algorithms can determine this level of information but nothing more. To support this, some tool developers have implemented a higher level roll-up capability as part of their user interface. That capability incorporates proprietary categorizations of network functionality and reflects the developer's perspective on discoverable assets.

[6] Some of the material in this section comes from *Common Platform Enumeration Technical Use Case Analysis,* The MITRE Corporation, November 2008. Cf. http://cpe.mitre.org/files/cpe_technical_use_cases.

A.3 Forensic Analysis/System Architecture Use Case

In the forensic analysis technical use case, tools are looking to tag things that are of interest to the forensic analysis being done. This need is driven by the fact that information about a specific vulnerability needs to be associated with the "thing" that it applies to. Unfortunately, many of the "things" that have vulnerabilities are artifacts or components contained within software products and are not products in and of themselves. Examples include drivers and individual DLL files. Historically, CPE has deliberately limited its scope to focus on naming "whole" products, as opposed to product parts or components.

A.4 IT Management Use Case

In the IT management view, platforms play functional roles (e.g., server). Some IT managers have expressed the desire to have lower-level CPE names roll up of somehow link to these functional roles. This is currently outside the scope of CPE. Unfortunately, the naming conventions for functional roles do not align with the current CPE convention for naming software and hardware products, which is based on the "who produced it?" perspective, not the "what is it used for?" perspective.

Appendix B—Change Log

Release 0 – 9 June 2010

- Initial draft specification released to the CPE community as a read ahead for the CPE Developer Days Workshop

Release 1 – 23 June 2010

- Minor edits to audience description.
- Minor editorial changes throughout the document.
- In section on Conformance, added a requirement that claims of conformance be made explicit in product documentation. Modified the third clause to allow conformers to "produce **and/or** consume", that is, an "and" became an "and/or", since some applications only need to produce and others only need to consume. Relaxed the requirement to consume legacy CPE names from a MUST to a SHOULD, since some applications may have no need to consume legacy content.
- Added an ABNF grammar to define character strings permitted as attribute values in WFNs.
- Switched to using the words/phrases "to quote" and "quoting" in place of "to escape" and "escaping" when referring to use of the escape character, to be more consistent with standard regular expression usage.
- Removed all mention of and support for the logical value UNKNOWN.
- Clarified the view that the logical value NA should also be used if an attribute value is assessed to be null.